Machine Learning HW 3 Report

組員：0516025 張智閔、0516032 邱繼聖、0516049 吳柏劭、0516215 林亮穎、0516220 李元毓

1. What environments the members are using?

OS: Window 10

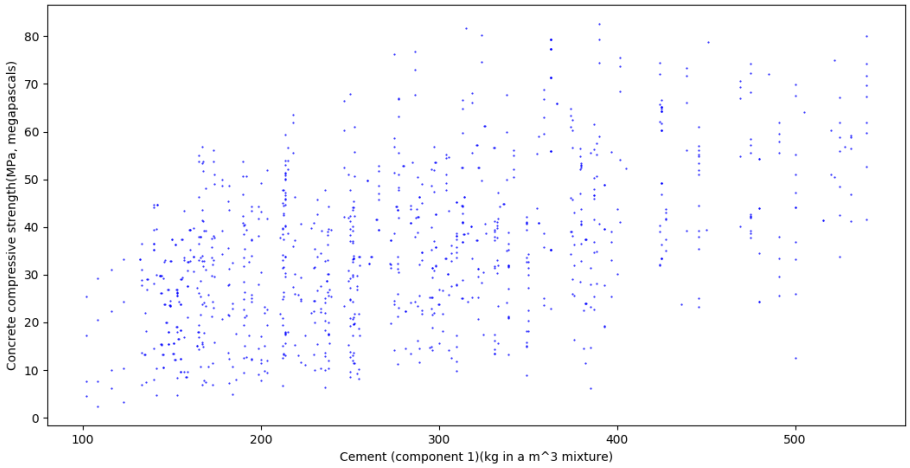
Language: Python

Packages:

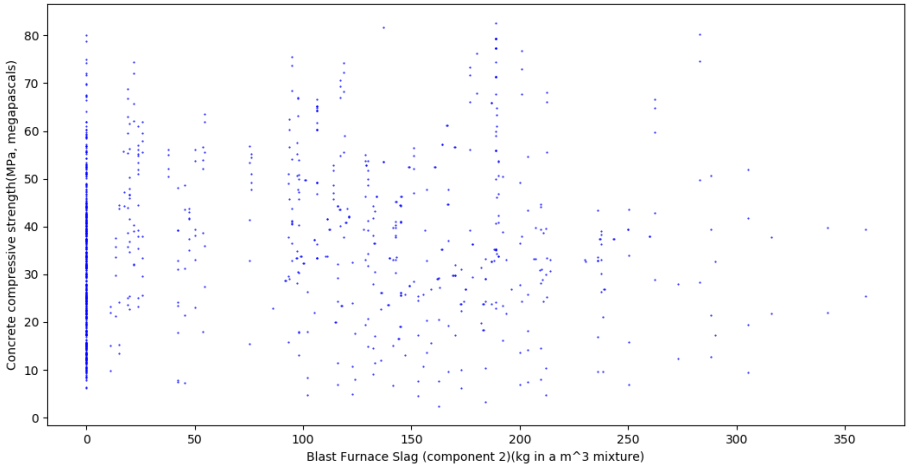
(1) numpy	1.15.2
(2) pandas	0.23.4
(3) scikit-learn	0.20.0
(4) matplotlib	3.0.0

2. Visualization of all the features with the target

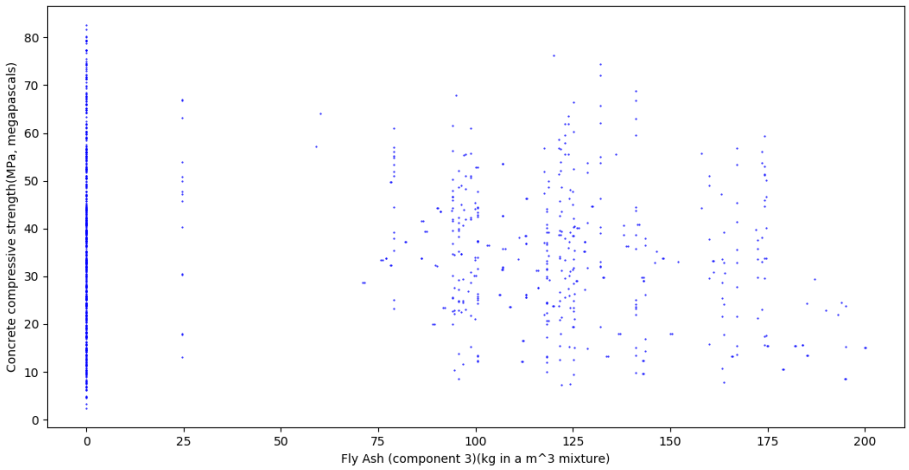
x--Cement y--Concrete compressive strength



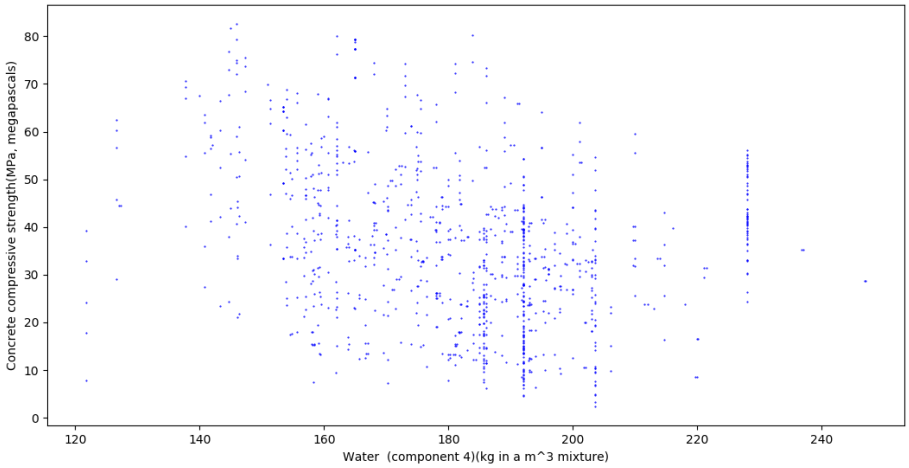
x-- Blast Furnace Slag y--Concrete compressive strength



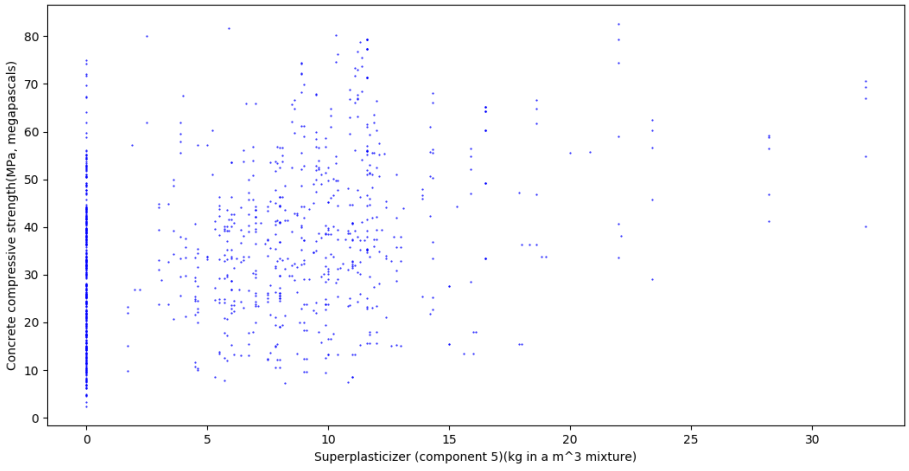
x-- Fly Ash y--Concrete compressive strength



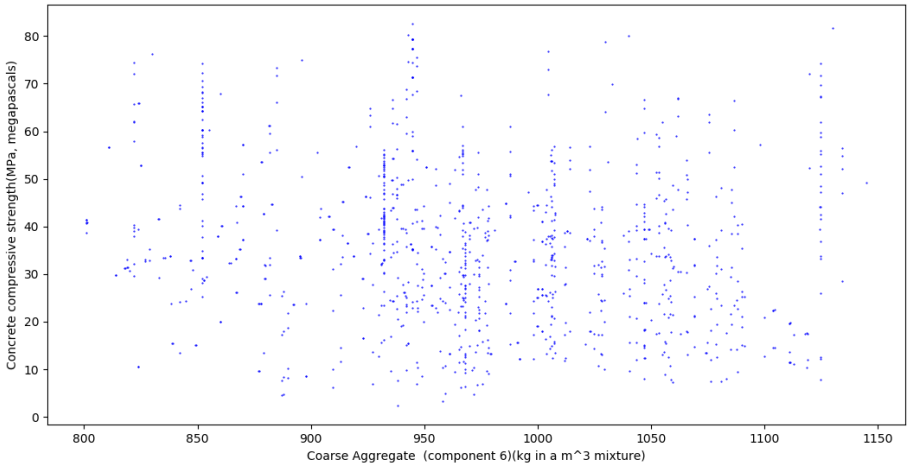
x-- Water y--Concrete compressive strength



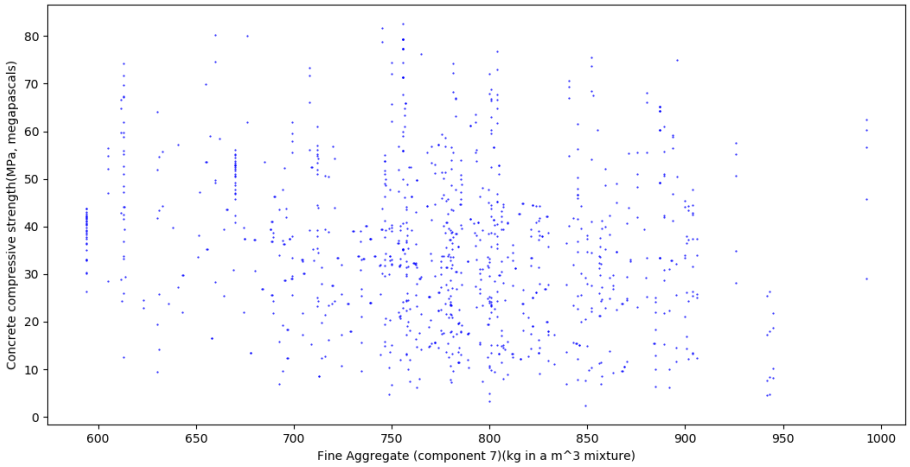
x-- Superplasticizer y--Concrete compressive strength



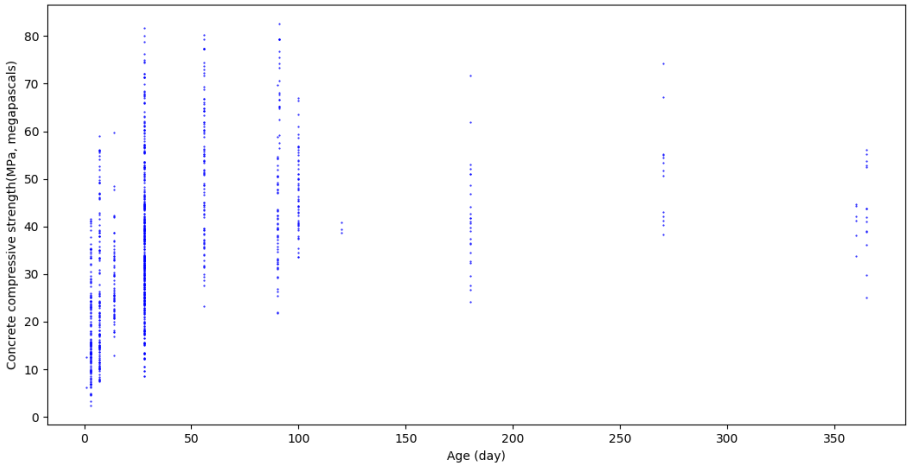
x-- Coarse Aggregate y--Concrete compressive strength



x-- Fine Aggregate y--Concrete compressive strength



x-- Age y--Concrete compressive strength



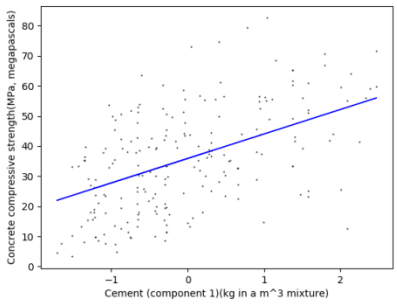
3. The code, graph, r2_score, weight and bias for problem 1

```
dataset = pd.read_csv('Concrete_Data.csv')
dataset_df = pd.DataFrame(dataset)

n_data = 1030

attrs = dataset.columns
attrs = attrs[:len(attrs)-1]

dataset_df = dataset_df.sample(n = n_data).reset_index(drop = True)

for column in attrs:
    print(column)

    dataset_X = dataset_df[column].values
    dataset_X = np.reshape(dataset_X, (len(dataset_X), 1))
    dataset_X = dataset_X.astype(float)
    dataset_X = preprocessing.scale(dataset_X)

    dataset_Y = dataset_df['Concrete compressive strength(MPa, megapascals)
'].values
    dataset_Y = np.reshape(dataset_Y, (len(dataset_Y), 1))

    dataset_X_test  = dataset_X[int(-n_data*0.2) : ]
    dataset_X_train = dataset_X[ : int(-n_data*0.2)]
    dataset_Y_test  = dataset_Y[int(-n_data*0.2) : ]
    dataset_Y_train = dataset_Y[ : int(-n_data*0.2)]

    regr = linear_model.LinearRegression()
    regr.fit(dataset_X_train, dataset_Y_train)

    dataset_Y_pred = regr.predict(dataset_X_test)
     print('Weight: ', regr.coef_)
    print('Bias: ', regr.intercept_)
    print("Mean squared error: %.2f" % mean_squared_error(dataset_Y_test,
dataset_Y_pred))
    print('R2 score: %.2f\n' % r2_score(dataset_Y_test, dataset_Y_pred))
```

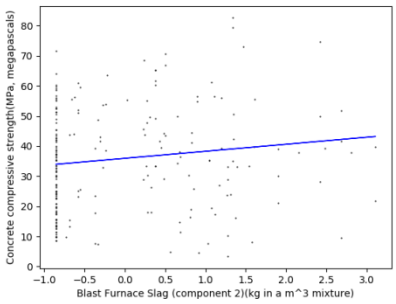
將資料讀入後利用 sklearn 內建的 LinearRegression function 進行分析

在這邊我們將所有的 features 都作了 normalization，使得每個 feature 的平均值為 0 標準差為 1，這樣做是為了之後實作 gradient descent 的時候，learning rate 跟 iteration count 可以不用取的那麼大。(Problem 2 , 3, 4 同樣也對 features 都作了 normalization)

Cement
Weight: 8.12266927
Bias: 35.89714205
R2_score: 0.28 (most relative)

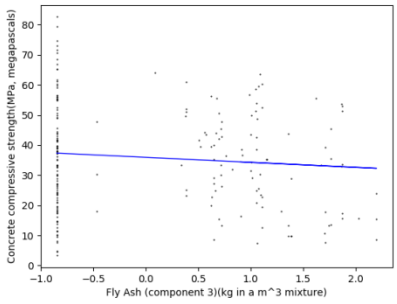


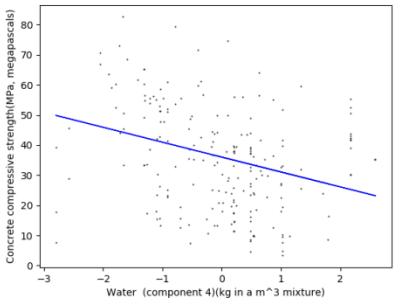
Blast Furnace Slag
Weight: 2.33483699
Bias: 35.9546199
R2_score: 0.01



Fly Ash
Weight: -1.65915058
Bias: 35.942211
R2_score: 0.01

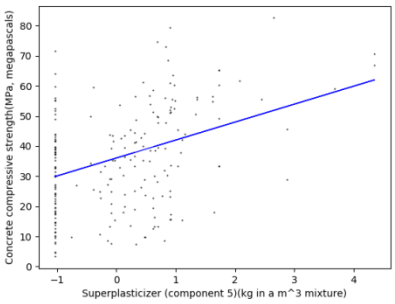
Water
Weight: -4.94666936
Bias: 36.02561127
R2_score: 0. 08

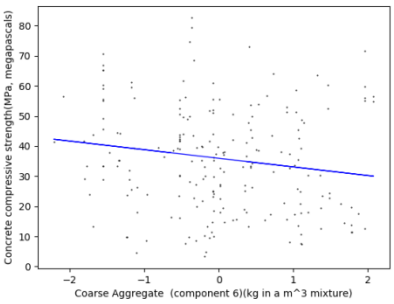




Superplasticizer
Weight: 5.96125137
Bias: 36.0575923
R2_score: 0.17

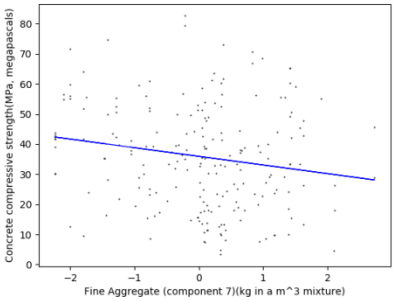
Coarse Aggregate
Weight: -2.86218863
Bias: 35.95651227
R2_score: 0.02

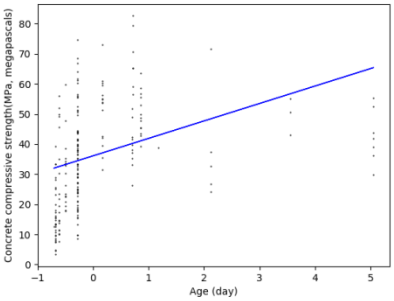




Fine Aggregate
Weight: -2.86776344
Bias: 35.93586509
R2_score: 0.02

Age
Weight: 5.79482615
Bias: 36.10255938
R2_score: 0.10





4. The code, graph, r2_score, weight and bias for problem 2

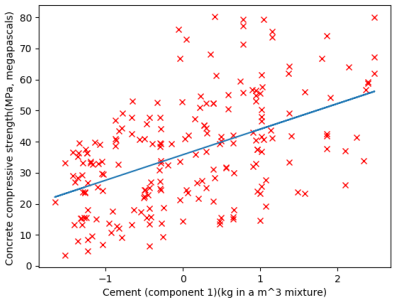
```
def CostFunc(X, Y, w):
    m = len(Y)
    return (1.0/2.0) * (1.0/m) * sum((np.dot(X, w) - Y) ** 2)

def GradientDescent(X, Y, w, learning_rate, itr_limit):
    m = len(Y)
    n = len(w)
    tmp = w[:]

    for itr in range(itr_limit):
        for i in range(n):
            tmp[i] = tmp[i] - learning_rate * (1/m) *
sum( np.dot(np.transpose((np.dot(X, w) - Y)), X[:, i]) )
        w = tmp[:]
```

```
    return w

if __name__ == '__main__':
    dataset = pd.read_csv('Concrete_Data.csv')
    dataset_df = pd.DataFrame(dataset)

    attrs = dataset.columns
    attrs = attrs[:len(attrs)-1]

    n_data = 1030
    dataset_df = dataset_df.sample(n = n_data).reset_index(drop = True)

    for column in attrs:
        dataset_X = dataset_df[column].values
        dataset_X = np.reshape(dataset_X, (len(dataset_X), 1))
        dataset_X = dataset_X.astype(float)
        dataset_X = preprocessing.scale(dataset_X)

        dataset_Y = dataset_df['Concrete compressive strength(MPa,
megapascals) '].values
        dataset_Y = np.reshape(dataset_Y, (len(dataset_Y), 1))

        X_test  = dataset_X[int(-n_data*0.2) : ]
        X_train = dataset_X[ : int(-n_data*0.2)]
        Y_test  = dataset_Y[int(-n_data*0.2) : ]
        Y_train = dataset_Y[ : int(-n_data*0.2)]

        one_train = np.ones((len(X_train), 1))
        X_train_padded = np.concatenate((one_train, X_train), axis=1)

        w = np.array([[0.2], [0.2]])
        iterations = 100000
        alpha = 0.0001

        print(column)

        w = GradientDescent(X_train_padded, Y_train, w, alpha, iterations)
        print("Bias:   %f" %w[0])
```

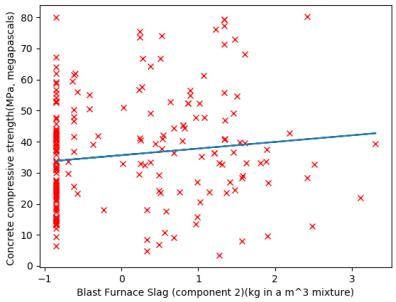
```
    print("Weight: %f" %w[1])

    one_test = np.ones((len(X_test), 1))
    X_test_padded = np.concatenate((one_test, X_test), axis=1)
    print("Mean squared error: %f" %CostFunc(X_test_padded, Y_test, w))

    Y_predict = np.dot(X_test_padded, w)
    print('R2 score: %.2f\n' % r2_score(Y_test, Y_predict))
```

自己實作 Gradient Descent，每次 iteration 都對 weight 及 bias 做調整，進行 100000 次後終止迴圈，取這時的 weight 和 bias 作為最終結果

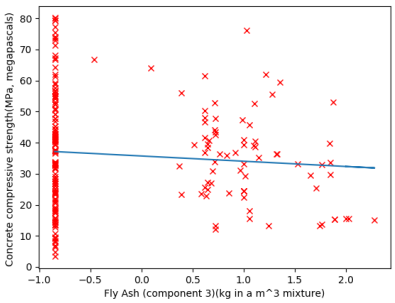
Cement
Weight: 8.236693
Bias: 35.795857
R2_score: 0.26 (most relative)



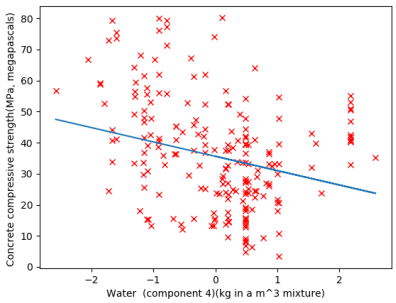
Blast Furnace Slag
Weight: 2.130658
Bias: 35.656680
R2_score: 0.02



Fly Ash
Weight: -1.687074
Bias: 35.711575
R2_score: 0.01



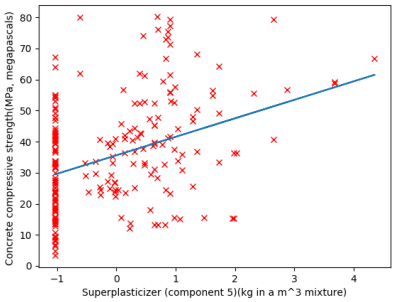
Water
Weight: -4.620336
Bias: 35.610997
R2_score: 0.10



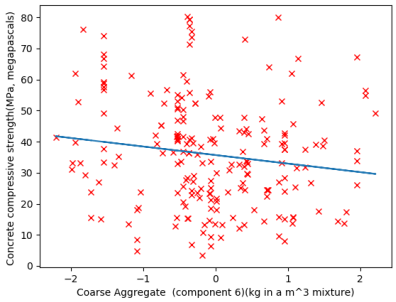
Superplasticizer
Weight: 5.953609
Bias: 35.616071

Coarse Aggregate
Weight: -2.745940
Bias: 35.688620

R2_score: 0.17



R2_score: 0.02

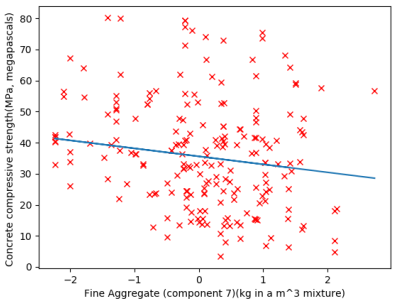


Fine Aggregate

Weight: -2.572224

Bias: 35.590758

R2_score: 0.04

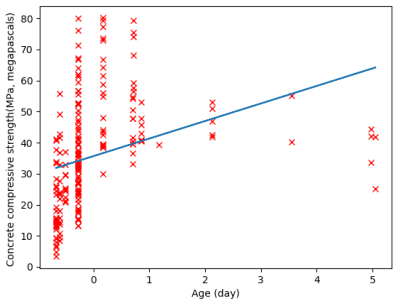


Age

Weight: 5.648707

Bias: 35.669246

R2_score: 0.08



5. Compare Problem1 and Problem2

Problem 1 和 Problem 2 所得出的結果十分相近，經過多次測試 Cement 都是相關度最高的 feature, 次之則是 Water, Superplasticizer 及 Age

Problem 1 和 Problem 2 最大差異在於執行速度，sklearn 中內建的 function 有經過相當程度的優化，以提升效率，但我們自己做的如果要加快速度，可能得犧牲準確度，減少 Gradient Descent 執行的次數

在 Minimum Square Error 的部分，Problem2 的會比 Problem 1 的小將近一半，這是因為 Problem2 在設計 cost function 時，我們是參照講義上的公式有在前面乘上 1/2，所以在計算 MSE 的時候，會和內建函式算出的 MSE 差上 1/2。(Problem 2 3 4 的 MSE 皆是如此)

6. The code, MSE, and the r2_score for problem 3

```
def ComputeCost(X, y, theta):
    m = y.shape[0]
    C = X.dot(theta) - y
    J = (C.T.dot(C)) / (2 * m)
    return J

def GradientDescent(X, y, theta, alpha, max_itrs):
    m = y.shape[0]

    for itr in range(max_itrs):
        theta = theta - (alpha / m) * (X.T.dot(X.dot(theta) - y))
    return theta

def GradientDescent_wj(X, y, theta, alpha, max_itrs):
    m = y.shape[0]
    n = theta.shape[0]

    for itr in range(max_itrs):
        for i in range(n):
            Xj = X[:, i]
            theta[i] = theta[i] - (alpha / m) * (Xj.T.dot(X.dot(theta) - y))
    return theta

dataset = pd.read_csv('Concrete_Data.csv')
dataset_df = pd.DataFrame(dataset)

n_data = 1030

attrs = dataset.columns
attrs = attrs[:len(attrs)-1]

dataset_X = dataset_df.drop(['Concrete compressive strength(MPa,
megapascals) '], axis = 1).values
dataset_X = preprocessing.scale(dataset_X)

dataset_Y = dataset_df['Concrete compressive strength(MPa, megapascals)
'].values
```

```
dataset_Y = np.reshape(dataset_Y, (-1, 1))

dataset_X_train, dataset_X_test, dataset_Y_train, dataset_Y_test =
train_test_split(dataset_X, dataset_Y, test_size = 0.2)

dataset_X_train = np.hstack([dataset_X_train,
np.ones((dataset_X_train.shape[0], 1))])
dataset_X_test = np.hstack([dataset_X_test,
np.ones((dataset_X_test.shape[0], 1))])

iterations = 1000
alpha = 0.1

w = np.zeros((dataset_X_train.shape[1], 1))
J = ComputeCost(dataset_X_train, dataset_Y_train, w)
print(J)

print('update w')
print('w1~w8 + Bias:')
w = GradientDescent(dataset_X_train, dataset_Y_train, w, alpha, iterations)
print(w)
print('MSE:')
J = ComputeCost(dataset_X_test, dataset_Y_test, w)
print(J[0][0])
dataset_Y_pred = dataset_X_test.dot(w)
print('R2 score:')
print(r2_score(dataset_Y_test, dataset_Y_pred))

print('------------------------')

w = np.zeros((dataset_X_train.shape[1], 1))
print('update wj')
print('w1~w8 + Bias:')
w = GradientDescent_wj(dataset_X_train, dataset_Y_train, w, alpha,
iterations)
print(w)
print('MSE:')
J = ComputeCost(dataset_X_test, dataset_Y_test, w)
```

```
print(J[0][0])
dataset_Y_pred = dataset_X_test.dot(w)
print('R2 score:')
print(r2_score(dataset_Y_test, dataset_Y_pred))
```

(1)each iteration only update wj

以 GradientDescent_wj()實作每個 weight 個別更新的方法

每項 feature 都跑過一次 wj = wj + αwj'，且過程中直接將 wj 做更新，所有 feature 和 bias 都做過後便完成一次 iteration，結果:

MSE: 56.90626217916698

R2_score: 0.5554463411798324

(2)each iteration updates w

以 GradientDescent()實作將 weight 同步更新的方法

利用上個 iteration 的 weight 及 bias 計算出 error 後，把所有 weight 和 bais 一起更新 w = w + αw'，結果:

MSE: 56.889181662408056

R2_score: 0.5555797747586503

7. Compare the performance between two different update method.

each iteration only update wj

計算完 wj'後馬上對 wj 做更新，每次計算 wj'都是用新的 w

each iteration updates w

算完 w'後再把 w 更新，每次計算 wj'都是用上次 iteration 後的 w

雖然計算完 wj'後馬上對 wj 做更新，每次計算 wj'都是用新的 w 是錯誤的作法，但實際上在測試中，兩種方法的差距很小，不過相較於每次只更新 wj，整個 w 一起更新不但是正確的且方便得多，因為可以直接利用矩陣運算對 w 做更新，能少做一層 for 迴圈

8. The code, MSE, and the r2_score for problem 4

```
def CostFunc(X, Y, w):
    m = len(Y)
    return (1.0/2.0) * (1.0/m) * sum((np.dot(X, w) - Y) ** 2)

def GradientDescent(X, Y, w, learning_rate, itr_limit, w_degree):
    m = len(Y)
    n = len(w)
    tmp = w[:]
```

```
    for itr in range(itr_limit):
        for i in range(n):
            Xi_power = X[:, i] ** i
            Xi_power = Xi_power.reshape(m, 1)
            tmp[i] = tmp[i] - learning_rate * (1/m) *
sum( np.dot(np.transpose((np.dot(X, w) - Y)), Xi_power) )
        w = tmp[:]
    return w

if __name__ == '__main__':
    dataset = pd.read_csv('Concrete_Data.csv')
    dataset_df = pd.DataFrame(dataset)

    attrs = dataset.columns
    attrs = attrs[:len(attrs)-1]
    n_data = 1030

    for column in attrs:
        pass

    dataset_X = dataset_df['Cement (component 1)(kg in a m^3
mixture)'].values
    dataset_X = np.reshape(dataset_X, (len(dataset_X), 1))
    dataset_X = dataset_X.astype(float)
    dataset_X = preprocessing.scale(dataset_X)

    dataset_Y = dataset_df['Concrete compressive strength(MPa, megapascals)
'].values
    dataset_Y = np.reshape(dataset_Y, (len(dataset_Y), 1))

    poly_degree = 2
    poly = preprocessing.PolynomialFeatures(degree=poly_degree)
    poly_x = poly.fit_transform(dataset_X)

    X_test_original  = dataset_X[int(-n_data*0.2) : ]
    X_test  = poly_x[int(-n_data*0.2) : ]
    X_train = poly_x[ : int(-n_data*0.2)]
    Y_test  = dataset_Y[int(-n_data*0.2) : ]
```

```
Y_train = dataset_Y[ : int(-n_data*0.2)]

w = np.array([0.5]*(poly_degree+1))
w = w.reshape(poly_degree+1, 1)
iterations = 100000
alpha = 0.0001

w = GradientDescent(X_train, Y_train, w, alpha, iterations, poly_degree)
print("Bias:   %f" %w[0])
print("Weight: ", end='')
print(w[1:])

print("Mean squared error: %f" %CostFunc(X_test, Y_test, w))

Y_predict = np.dot(X_test, w)
print('R2 score: %.2f\n' % r2_score(Y_test, Y_predict))
```

在這一題，我們對單一個 feature 作了 polynomial regression

因為 Cement 與 output 的相關度最高

所以我們嘗試令 x = {1, x1, $x1^2$}做非線性的迴歸分析，結果:

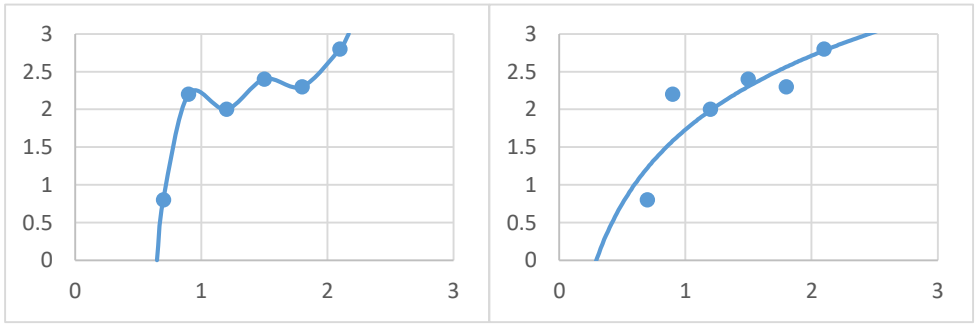
MSE: 45.068461　　R2_score: 0.41

9. Answer the question

a. What is overfitting?

過於追求 model 的預測完美對應 training set，使得實際導入 testing set 後的預測效果不佳，以下是示意圖

Overfitting　　Better one



以本次作業來說的話，在 Problem4 polynomial regression 時，若是硬要用過高次方的函數來 train 的話，就有可能產生 over fitting，反而使得 r2 score 降低。

b. Stochastic gradient descent is also a kind of gradient descent, what is the benefit of using SGD?

Stochastic gradient descent 在每次 iteration 中只使用單一個 training example

來計算 error 而非整個 training set。雖然這會造成它沒辦法收斂到最佳解，但結果也十分接近。當資料量很大時，Stochastic gradient descent 比較方便計算，不須進行龐大的矩陣運算，節省的時間和計算量讓這誤差可以被接受。

c. Why the different initial value to GD model may cause different result?

在作多變數的 regression 時，gradient descent 出來的可能會有多個 local minimum，不同的 initial value 可能使得 Gradient Descent 找到不同的 local minimum，因此會產生其他結果。

d. What is the bad learning rate? What problem will happen if we use it?

過大或過小的 learning rate，讓 model 無法逼近最佳解或者花費太多次 iteration 才得到最佳解。太大的 learning rate 會使每次的 error 修正過多，讓得到的結果在最佳解附近震盪，或者是直接發散，無法收斂到 local minimum；太小的 learning rate 使 error 的修正量過小，得經過更多次的 iteration 以達到最佳解。

e. After finishing this homework, what have you learned, what problems you encountered, and how the problems were solved?

我們學會如何讓程式從資料中自行找出最佳的迴歸函式。

這次的作業中，比較苦惱的問題是在如何找到適當的 learning rate 值。當設定過大，造成 model 無法趨近於最佳解。而設定過小則造成需要足夠多次的 iteration 才能收斂到 local minimum。我們當初在實作時社的 learning rate 太小而且 iteration 跑的次數不夠多，使得在每一次 iteration 時，數值的改變量非常小，讓我誤以為已經收斂了，但是收斂到的值卻跟 built-in function 的結果相差很多，我還花了很多時間檢查 model 有沒有寫錯，最後才發現是 iteration 跑的次數不夠多，只要調一下參數就好。

另外，當初因為 learning rate 設的小，iteration 次數必須足夠多才能獲得比較接近的答案，使得每次 training 都要花上很多時間。後來我們查了資料後，將所有 features 都作了 normalization，這樣 feature 對 regression 係數比較敏感，使得 learning rate 可以設的大一點，iteration 也不需要跑那麼多次，就可收斂到 local minimum，可有效改善程式的效能跟執行時間。

10.Bonus

```
Bias:   41.605010
Weight: [[  9.90433577  -1.25076431   0.98339479   8.5220554   -0.36382391
   -0.35999326   4.88778773   6.32509285  -3.51245707  -6.58869001
    0.63990682   0.90461829   0.66176319  -2.75654873   0.49780949
   -0.37631471   0.10672448   0.30884364  -0.89583222  -0.71270292
    0.60566724  20.89003461 -11.75571672   1.58442675]]
Mean squared error: 20.336177
R2 score: 0.8711
```

在 Bonus 的部分，我將 Problem 4 的 model 改寫為多變數版本的，也就是作多變數的 polynomial regression。

參數的部分，我將所有的 features 都丟入 gredient descent，回歸方程式的最高次方設為 3，learning rate = 0.007，iterations = 100000，就跑出上圖 R2 score = 0.8711 > 0.87 的結果。